AFOSR-TR-97-0468

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 15 April 97	3. REPORT TYPE AND DATES COVERED Final Technical Report 15 Aug 93-14 Feb 97

4. TITLE AND SUBTITLE
Final Technical Report on Coherent Transient Systems Development

5. FUNDING NUMBERS
61102F
2305/BS

6. AUTHORS
Wm. Randall Babbitt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Washington
Department of Electrical Engineering
Box 352500
Seattle, WA 98195-2500

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research/NE
110 Duncan Avenue, Suite B115
Bolling AFB, DC 20332

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-93-1-0513

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The main objective of the program was to determine if and where coherent transient technology can be used in future memory and signal processing devices, to develop techniques for implementing practical devices, and to demonstrate the feasibility of the applicable devices. The approach encompassed six main areas: 1) analysis and simulation, 2) evaluation, 3) experimental validation of theory, 4) material exploration and development, 5) feasibility demonstrations, and 6) collaborative interactions with material growers and system designers. Accomplishments were made in the following areas: coherent transient continuous optical processing, spatial-spectral holographic optical routing and processing, coherent transient header/data isolation, relationships between material properties and system bandwidth, chirped reference pulse techniques for coherent transient processors, compensation for the deleterious effects of homogeneous dephasing, true-time delay regeneration and processing, enhanced output efficiencies, spatial spectral holographic frequency division multiplexing, and optical coherent transient signal simulation.

14. SUBJECT TERMS Optical coherent transients, optical memories, optical processors, optical routing, freuquency division multiplexing, true-time-delay, phase-array antenna, triple-product correlator, temporal convolution, continuous signal processing, optical Bloch equations, homogeneous dephasing, chirped reference pulses, spectral holography

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

STANDARD FORM 298 (Rev 2-89)
Prescribed by ANSI Std 239-18
298-102

Final Technical Report

on

Coherent Transient Systems Development

Grant Number

F49620-93-1-0513

Submitted to:

Dr. Alan E. Craig
Air Force Office of Scientific Research/NE
110 Duncan Ave., Suite B-115
Bolling AFB, DC 20332-0001

By

Wm. Randall Babbitt
Principal Investigator
Department of Electrical Engineering, 352500
University of Washington
Seattle, Washington 98195-2500

Covering 8/15/93 to 2/14/97

EXECUTIVESUMMARY

The main objective of the program (F49620-93-1-0513) was to determine if and where coherent transient technology can be used in future memory and signal processing devices, to develop techniques for implementing practical devices, and to demonstrate the feasibility of the applicable devices. The approach encompassed six main areas: 1) analysis and simulation, 2) evaluation, 3) experimental validation of theory, 4) material exploration and development, 5) feasibility demonstrations, and 6) collaborative interactions with material growers and system designers.

Almost all the program's findings have been or will be published in archival journals or proceedings. The program resulted in 9 published and 4 to be submitted journal papers, 7 refereed conference publications, 5 invited talks, several other conference and workshop presentations, and 2 patents. For those findings describe elsewhere, brief summaries of the work accomplished are given along with references to the relevant published articles or proceedings. The work accomplished under this program was augmented by an ASSERT grant F49620-95-1-0468 and DURIP equipment F49620-95-1-0477.

The significant accomplishments of the program include:

1) a theoretical evaluation of the performance of the coherent transient continuous optical processor.
2) the demonstration of the continuous correlator showing that correlation and convolution of signals can be done well beyond the homogeneous decay time limit.
3) an analysis of the performance of coherent transient memory systems based on physical limits.
4) the development of a novel concept in optical routing and processing that combines the temporal processing capabilities of coherent transients with the spatial processing of volume holography.
5) the proposal and demonstration of novel techniques for header/data isolation based on spatial-spectral holography.
6) the analytical development of scaling relations that relate material properties to the system data bandwidth.
7) the demonstration of the replacement of the single brief reference pulse used in coherent transient processors with two long, frequency chirped pulses.
8) the development and demonstration of techniques to compensate for the deleterious effects of homogeneous dephasing in coherent transient optical memories and processors.
9) The development and demonstration of the application of spatial-spectral holography to true-time delay processing for phased array radar transmitters and receivers.
10) An analysis of the enhanced efficiencies that are achievable in inverted media and in optically thick media.
11) The demonstration of the use of spatial spectral holography for frequency division multiplexed communication, interconnection, and routing.
12) Further development of an optical coherent transient signal simulator.

TECHNICAL SUMMARY OF WORK ACCOMPLISHED

Performance Evaluation of the Coherent Transient Continuous Optical Correlator

In collaboration with John A. Bell at Boeing, a theoretical evaluation of the performance of the coherent transient continuous optical processor was completed. The analysis considered the effects of coherent saturation, incoherent saturation, gating efficiency, pattern pulse compression, homogeneous decay, cooperative emission, spatial beam overlap, spatial crosstalk between adjacent beams, diffraction, and local thermal heating. The main inputs to the analysis are the wavelength, the inhomogeneous linewidth, the homogeneous linewidth, the number density of absorbing centers, gating efficiency, index of refraction, and detector efficiency. Additional inputs involve the spectral and spatial characteristics of the pattern and data streams and the acceptable level of nonlinearities. Given the number of detected photons per bit needed to achieve a desired signal to noise, the analysis yields the required spot size and maximum pattern density for a given set of input parameters. For example, for a material with reasonably obtainable characteristics the correlator's expected data rate, time-bandwidth product, and pattern storage-density are respectively 5 gigahertz, 16000, and 110,000 patterns per square centimeter. Higher performance depends on developing higher performance materials. The calculation also yields the optimal oscillator strength and path length through the crystal. The work was published in Applied Optics **33**, 1538-1548 (1994) and resulted in U. S. Patent No. 5,239,548.

Coherent Transient Continuous Optical Correlator Demonstration

In collaboration with Miao Zhu and C. Michael Jefferson at IBM, it was demonstrated that correlation and convolution can be done with coherent transients well beyond the homogeneous decay time limit. The basic concept that was demonstrated was that once the spectral interference grating is stored in the ground state, it can be continuously interrogated provided the transition is not saturated. This allows the material to act as a passive spatial-spectral filter with application in signal processing, database memory, and routing applications. In the demonstration, a 3120-bit long phase encoded data stream was correlated with a 13-bit phase encoded barker code. Even though the data stream's duration exceeded the upper state lifetime of the excited transition (Thulium doped Yttrium Aluminum Oxide), the output signal intensity maintained level and the output's fidelity was excellent. This demonstrates ability to perform continuous real-time processing of phase and amplitude encoded optical data streams using coherent transient techniques. The results were presented as an invited talk at Spectral Hole-Burning and Related Spectroscopies: Science and Applications, August 24-25 1994 and published in Optics Letters **20**, 2514-2516 (1995).

Physical Limits Of Coherent Transient Memories With Two-Dimensionall Addressing

In collaboration with Thomas W. Mossberg of the University of Oregon, an analysis was made of the performance of coherent transient memory systems in which multiple bits are temporally stored at each spatially addressed laser spot. This approach differed from the approach used in an earlier paper(W. R. Babbitt, Proc. SPIE 2026, 532-542 (1993)), which is similar to the approach used in the evaluation of the continuous correlator. In the earlier memory and continuous correlator analyses, a particular implementation and set of performance criteria were assumed, which yielded an overall efficiency of the process that was particular to the chosen inputs. In this paper, no particular implementation was assumed. Instead, the overall efficiency of the process is an input parameter. Based on consideration of the physical limits involved in the two-dimensional storage process, limits were placed on the maximum obtainable areal storage densities for coherent transient memories. For a material that allows one million bits per laser spot, we fold a 20,000-fold increase in areal storage over traditional 2D optical memories. These are the physical limitations and known practical considerations and implementations may not enable these densities to be obtain. However, novel implementations are continually being developed that push the technology closer to the physical limits. In addition, higher performance can be obtained using volume holographic storage capabilities of coherent transients. The physical limits of coherent transient volume storage have yet to be analyzed, but initial calculations indicate significantly higher areal densities can be achieved with relaxed absorber concentration requirements. The results of the physical limits of quasi two-dimensional storage were published in JOSA B 11, 1948-1953 (1994).

Spatial-Spectral Holographic Routing Of Optical Beams

In collaboration with Thomas W. Mossberg of the University of Oregon, a novel concept in optical routing and processing that combines the temporal processing capabilities of coherent transients with the spatial processing of volume holography. This new concept, referred to as spatial-spectral holography, has applications in code-division multiplexed routing, packet header recognition, header/data isolation, and contention resolution. A patent application has been filed jointly. The work resulted in several conference presentations and in the paper "Spatial routing of optical beams through time-domain spatial-spectral filtering" (Optics Letters **20**, 910-912).

Optical Coherent Transient Header/Data Isolation

Novel techniques for header/data isolation based on spatial-spectral holography were proposed and demonstrated. Using these techniques, ultra-high bandwidth optical information stream can be parsed into two segments. Either segment can be recalled separately (isolated from the other). As an example, an information packet with a header and a data segment can be parsed. The isolated header can be processed and then the stored data segment can be routed or processed. The novel header and data isolation techniques and their experimental demonstrations were presented at HBRS'96 and in Optics Letters 21, 71 (1996).

Scaling Of Material Properties With Data Bandwidth

One important finding in the evaluations of coherent transient systems is that as the bandwidth of devices are pushed to higher bandwidths, the material properties that relate to rates must also be proportionally increased. Most significantly, the optimal oscillator strength scales with the data bandwidth. This may eliminate rare earth f-f transition from consideration in terahertz devices. This finding assumes a constant number of detected photons per bit to achieve a desired bit error rate. As bandwidths increase, a higher number of photons per bit may be required. Under these conditions, the bandwidth and material rates won't scale proportionately, but the trend will still be towards higher oscillator strengths. These findings have been presented and discussed at the several workshops on applications of spectral holeburning.

Chirped Reference Pulses In Coherent Transient Processing

The replacement of the single brief reference pulse used in coherent transient processors with two long, frequency chirped pulses was demonstrated. Eliminating the intense brief pulse requirement makes programming coherent transient processors considerably more practical. As proposed in the original patent on coherent transient processing, the first brief pulse in triple product correlation sequence could be replaced by two frequency chirped pulses. We have both analytically and experimentally shown this. We have also performed calculations that show that another previously suggested implementation will not work, but that three novel implementation we have proposed we work. One of these novel implementations was demonstrated as well as the other previously proposed implementation. The demonstration results were presented at OSA Annual Meeting 1994, Photonics in Computing II, and in Applied Optics 35, 278 (1996).

Compensation for homogeneous dephasing in coherent transient systems

Techniques to compensate for the deleterious effects of homogeneous dephasing in coherent transient optical memories and processors were developed and demonstrated. By tailoring the pattern pulses that program the coherent transient material, distortions of output signals can be significantly reduced, allowing time-bandwidth product or storage capacity of coherent transient devices to be increased by up to an order of magnitude. These techniques may aid in the practical implementation and enhanced performance of coherent transient routers, processors, and memories. Trade-offs with signal size must be considered in evaluating the overall performance of a system implementing homogeneous decay compensation. The techniques are described and demonstrated in the proceedings of CLEO'96 and HBRS'96 and in Optics Communications 128, 136 (1996).

Coherent transient true-time delay regenerators and processors

The application of spatial-spectral holography to true-time delay processing for phased array radar transmitters and receivers has been developed. Coherent transient true-time delays have the ability to simultaneously, and continuously process several hundred delays over a broad bandwidth with fine temporal resolution. This will enable wide angle (> 45 degrees) beamsteering of wide-bandwidth (> 10GHz) for multi-element (> 1000) arrays. Multicasting and simultaneous delay and temporal processing capabilities are additional features. The concepts and their demonstration have been presented at several conferences, including HBRS'96 and in Optics Letters 21, 1102 (1996). Various techniques for programming the coherent transient true-time delay regenerator using frequency chirped pulses are described in a manuscript being submitted to Optics Letters. The role of non-linearities in the material excitation of the delay timing is described in a manuscript being prepared for submission to Physics Review Letters.

Echoes in an Inverted Media and Optically Thick Media

The enhanced efficiencies that are achievable in inverted media has been shown via computer simulation. The most significant result is that data recall efficiencies much greater than unity are achievable in optically thick inverted media while maintaining high fidelity. Enhanced efficiencies are critical to making coherent transient devices practicable. The results for the inverted media were described at the HBRS'96 conference and in a paper published in Mol. Cryst. Liq. Cryst. **291**, 269 (1996). A comprehensive study of the efficiencies of data pulse recall in non-inverted optically thick media is being continued under our new AFOSR grant and a paper is being submitted to Phys. Rev. A. The most significant result here is that efficiencies approaching unity are achievable in even non-inverted media, which is counter to the conventional wisdom. Experiments to verify these results are in progress.

Spatial Spectral Holography Frequency Division Multiplexing

The proposed use of spatial spectral holography (SSH) for frequency division multiplexing (FDM) was demonstrated. In contrast to volume holographic FDM, where the efficiencies are inversely proportional the square of the number of channels, the efficiency of SSH FDM routing is independent of the number of channels. For a 1000 channel FDM system, the 10-25% channel efficiency of SSH FDM is extraordinary compared to efficiencies of on the order of (1/1000)2 per channel achievable with volume holography. With channel bandwidths over a gigahertz, SSH FDM could enable over terahertz communication links. In addition, the ability of SSH to couple any incident beam direction to any outgoing beam direction depending on the incident beams frequency may be important in SSH N x N routing applications. The concept is described in the proceedings of DEPOM'96 and CLEO'97. A manuscript describing a demonstration of the concept is being submitted to Optics Letters.

Coherent Transient Simulator

Several of the work done under this program have relied on a computer program which simulates the optical coherent transient signal produced from an arbitrary sequence of input excitation pulses. As the code for the simulator has not been published elsewhere, it is contained in appendix A.

The versatile simulator, begun under AFOSR contract F49620-91-C-0088 and improved under that current grant, integrates the optical Bloch equations and follows the evolution of the density matrix elements during and between excitation pulses. After the final excitation pulse, the density matrix elements contain the information necessary to calculate the resultant coherent transient response of the absorbing medium. Since the Bloch equations take into account nonlinear effects, the program is useful in studying the effect nonlinear excitation pulses. The program currently runs a Gateway 486-33MHz machine, but is written in C using the standard I/O interface and is thus directly portable to workstations or mainframe computers and could be incorporated in the core of channel modeling program used to evaluate the performance of memory and processing systems.

The coherent transient simulation program has been optimized for speed, flexibility, and input and output capabilities. The simulator accepts as input an arbitrary number of excitation pulses, including chirped pulses and arbitrarily long phase-, amplitude-, or frequency-encoded data pulses. The exact analytical transformation of the density matrix elements resulting from excitation by a square pulse of arbitrary duration, amplitude, center frequency, phase, and timing was calculated and incorporated into the program. This greatly enhanced the programs speed since integration of the matrix elements was no longer required. Chirped pulses of arbitrary duration, bandwidth, amplitude, center frequency, phase, and timing are calculated by integration of the optical Bloch equations. The response of the medium to other types of excitation pulses of arbitrary temporal structure can be calculated in the same way.

Input capabilities and calculation flexibility include options for 32-bit data pulses, gating pulses, sudden losses in coherence, and homogeneous decay. Data pulses up to 32-bits long can be entered as a single integer and can be amplitude or phase encoded, return to zero (RZ) or non-return to zero (NRZ). Data pulses longer than 32-bits are achieved by repeating this option indefinitely. The gating pulse option simulates the effects of a gating pulse by eliminating those atoms in the upper state at the time of the pulse, thus permanently modifying the inhomogeneous profile. The resultant modulations in the inhomogeneous profiles will alter the effect of excitation pulse nonlinearities compared to an unmodulated inhomogeneous profile. The ability to create a sudden loss of coherence aids in simulating absorbers with homogeneous decay times much shorter than the upper state lifetimes and in simulating the rejection of spurious outputs via spatial isolation (since the program does not do spatial integrations). Homogeneous decay can be optionally included in the standard Bloch equations for chirped or square pulses. The output signals resulting that would be observed with direct detection and coherent detection are calculated, as well as the resultant holeburning spectra. The graphical interface allows quick evaluation of the results. The graphical interface is achieved via a different program (Microsoft Excel) that is simultaneously resident with the coherent transient simulator. This separation maintains the quick portability of the simulator to other machines.

PUBLICATIONS

Refereed Journal Publications

W. R. Babbitt and J. A. Bell, "Coherent Transient Continuous Optical Processor," Applied Optics, 33, 1538 (1994).

W. R. Babbitt and T. W. Mossberg, "Quasi-two-dimensional time-domain color memories: process limitations and potentials," JOSA B **11**, 1948-1953, (1994).

W. R. Babbitt and T. W. Mossberg, "Spatial routing of optical beams through time-domain spatial-spectral filtering, " Optics Letters 20, 910-912 (1995).

M. Zhu, W.R. Babbitt, C. M. Jefferson, "Continuous coherent transient optical processing in a solid," Optics Letters **20**, 2514-2516 (1995).

K.D. Merkel and W.R. Babbitt, "Optical coherent transient header/data isolation technique," Optics Letters **21**, 71-73 (1996).

K.D. Merkel and W.R. Babbitt, "Coherent transient optical signal processing without brief pulses," Appl. Optics, **35**, 278-285 (1996).

K.D.Merkel and W.R.Babbitt, "Compensation for homogeneous dephasing in coherent transient optical memories and processors," Optics Communications, **128**, 136 (1996).

K. D. Merkel and W. R. Babbitt, "Optical coherent transient true-time delay regenerator," Optics Letters **21**, 1102-1104 (1996).

M. A. Azadeh and W. R. Babbitt, "Photon Echoes in Inverted Media," Mol. Cryst. Liq. Cryst. **291**, 269-276 (1996).

M. A. Azadeh and W. R. Babbitt, "Photon Echoes in Optically Thick Media," Submitted to Phys. Rev. A.

K. D. Merkel and W. R. Babbitt, "Coherent transient true-time delay programming using freuqency-chirped pulses" to be submitted to Optics Letters

W. R. Babbitt, "Spatial Spectral Holography Frequency Division Multiplexing," to be submitted to Optics Letters.

K. D. Merkel, M. Azadeh, and W. R. Babbitt, "Time shift of true-time delay regenerated signals in optical coherent transients stored with brief pulses", manuscript in preparation for submission to Phys. Rev. Lett.

Refereed Conference Publications

W. R. Babbitt and T. W. Mossberg, "Optical Beam Routing through Time-Domain Spatial-Spectral Filtering," Photonics in Switching, OSA Technical Digest Series Vol. 12 (OSA, Washington, DC, 1995), pp 129-131, Salt Lake City, Ut., March 15-17, 1995.

W. R. Babbitt and T. W. Mossberg, "Passive Optical Beam Routing through Temporal Spatial-Spectral Filtering, " in Conference on Lasers and Electro-Optics, 1995 Technical Digest Series (Optical Society of America, Washington, DC, 1995) Vol . 15, pp.86-87, Baltimore, Md., May 21-26, 1995.

K. D. Merkel and W.R. Babbitt, "Coherent transient optical signal processing without brief reference pulses," in Photonics in Computing II, Leo J. Irakliotis, Editor, 117-126 (1995), Fort Collins, Co., June 6-9, 1995.

K. D. Merkel and W. R. Babbitt, "Homogeneous dephasing compensation in coherent transient optical memories and processors," in Conference on Lasers and Electro-Optics, Vol. 9, 1996 OSA Technical Digest Series (Optical Society of America, Washington, DC, 1996) pp 197-198. Anaheim, Ca., June 2-7, 1996.

K. D. Merkel and W. R. Babbitt, "A coherent transient optical true-time delay generator for phased array radar," Poster presentation at The 5th Intl. Meeting on Hole Burning and Related Spectroscopies (HBRS '96), Brainerd, MN., Sept. 13-17, 1996.

K. D. Merkel and W. R. Babbitt, "Optical coherent transient header/data isolation technique," Poster presentation at The 5th International Meeting on Hole Burning and Related Spectroscopies (HBRS '96), Brainerd, Minn., September 13-17, 1996.

M. Azadeh and W. R. Babbitt, "Coherent Transients in Inverted Media," Post-deadline poster presentation at The 5th International Meeting on Hole Burning and Related Spectroscopies (HBRS '96), Brainerd, Minn., September 13-17, 1996.

Invited Conference Publications

Miao Zhu, C. Michael Jefferson, and W. R. Babbitt† († Invited Speaker), "Coherent Transient Continuous Optical Processing in a Solid," Spectral Hole-Burning and Related Spectroscopies: Science and Applications, **15**, 1994 OSA Technical Digest Series (Optical Society of America, Washington, DC, 1994) 392-395, Tokyo, Japan, Aug. 24-26, 1994.

W. R. Babbitt, "Routing of optical data through temporally encoded spatial-spectral gratings," OSA Annual Meeting 1995, Portland, OR, September 10-15, 1995.

K. D. Merkel and W. R. Babbitt† († Invited Speaker), "Coherent transient optical memories and processors with homogeneous dephasing compensation," The 5th Intl Meeting on Hole Burning and Related Spectroscopies (HBRS '96), Brainerd, MN September 13-17, 1996.

W. R. Babbitt, "Spatial-Spectral Holographic Memories, Processor, and Routers," in *Optics in Computing*, Vol. 8, 1997 OSA Technical Digest Series (OSA, Washington DC, 1997), pp172-174, Incline Village, NV, March 16-21, 1997.

W. R. Babbitt, "Spatial-Spectral Holographic Routing and Processing Devices", Conference of Lasers and Electro-Optic 1997, Baltimore, MD, May 18-23, 1997.

Non-Refereed Conferences Proceedings

K. D. Merkel and W.R. Babbitt, "Coherent transient optical correlator without brief reference pulses," (abstract only) OSA Annual Meeting 1994, (pg. 177), Dallas, TX , October 2-7, 1994.

Workshop Presentations and Lectures

W. R. Babbitt, "Spectral-Spatial Holographic Optical Storage and Processing," Montana State University, Dept. of Physics, Colloquium, May 5, 1995.

W. R. Babbitt, "PSHB System Analysis," Material Requirements for Persistent Spectral Hole Burning and Time-Domain Optical Storage and Processing, August 3-4, 1995, Bozeman, MT.

W. R. Babbitt, "Spatial-Spectral Holographic Network Devices," Workshop on "Multiple Wavelengths in Free-Space Optical Interconnects," Taos, NM, Feb. 4-7, 1996.

W. R. Babbitt, "Persistent Spectral Hole-burning Memories & Processors," Workshop on Data Encoding for Page-oriented Optical Memories (DEPOM'96), Phoenix, AZ, March 27-28, 1996.

W. R. Babbitt, "Novel Applications of Spatial-Spectral Holography. Status of Cryocooler Research," Workshop on Applications of Persistent Spectral Holeburning, Big Sky, MT, March 3-6, 1996.

W. R. Babbitt and M. Azadeh, "Coherent Transients in Inverted Media," Frontiers of Applications of Photospectral Holeburning Workshop, Big Sky, MT, Feb. 16-19, 1997.

Patents Issued and Filed

W. R. Babbitt and J. A. Bell, "An optical signal processor for processing continuous signal data," U. S. Patent No. 5,239,548 (August 24, 1993)

W. R. Babbitt and T. W. Mossberg, "Apparatus and methods for routing of optical beams via time-domain spatial-spectral filtering," U. S. Patent filed March 13, 1995.

Web Sites and Listserves Maintained

Persistent Spectral Holeburning Web Page and Bulletin Board listserve
Web site: http://weber.u.washington.edu/~rbabbitt/pshb.html
Listserve: pshb@ee.washington.edu

APPENDIX A

CODE FOR COHERENT TRANSIENT SIMULATOR

Some of the following routines were adapted from those published in the book Numerical Recipes in C by W H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Cambridge Unviersity Press, 1988. Not included here are routines that are extensively identical to those found in Numerical Recipes in C and one should refer to this book for the code for these routines.

BLOCH.C

```
/*** This program integrates the optical Bloch equations for a two-level
system where 2(PI)V(t)/h = omegaR(t-tstart)*cos(omega0*t +
         (chirp/2)*(t-tstart)^2 + phi0 + PM(t-tstart) )
  = omegaR(t-tstart)*cos(omega10*t + phi(t)),
  where
    detune = omega10-omegac
    detuneL= omega0-omegac
    detuneAL = omega10 - omega0 = detune - detuneL
    phi(t) = A + B*(t-tstart) + C*(t-tstart)^2 + PM(t-tstart)
    A     = phi0 - detuneAL*tstart
    B     = -detuneAL
    C     = chirp/2
    omega(t)= omega0 + chirp*(t-tstart) + d/dt(PM(t-tstart))
         = instantaneous frequency.
Density matrix elements:
rho00=Bloch[1], rho01real=Bloch[2], rho01imag=Bloch[3],population density=Bloch[0]
Pulse parameters that can be changed during runtime in set_values() are:
  omegaR, chirp, phi0,detuneL,tstart,tstop.
  detuneL is the laser detuning from center of the inhomogeneous profile.
  detune is the atoms detuning from the center of Inh. profile and varies
  from detune1 to detuneN where Ndetune is the number of detunings.
Two functions are needed in derivs:
  omegaR(t)=>omegaRt
  phi(t)=>phit.
The program integrate the OBE for Npulse pulses for Ndetune different
  detunings. The pulse parameters can be independently set. The data
  is appended to the specified output file after each pulse. FFT information is
  stored in file graphit.dat for graphing with IGOR.
Project includes:
bloch.c, complex.c, OBE_int.c,OBE_bs.c,OBE_rk.c,OBE_sq.c,OBE_fft.c,
gasdev.c, ran1.c,ran2.c, and nrutil.c
***/
#define maxNdetune 4096+1 /*** maximum number of detunings ***/
#define maxNN2 (2*(maxNdetune-1))
#define maxNpulse 10  /*** maximum number of pulses ***/
#define TINY  1.0e-30 /*** definition of a tiny value ***/
#define SQR(a) ((a)*(a))
#define  PI  3.14159265358979
#define  PI2 6.28318530717958
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <io.h>
#include <math.h>
#include "complex.h"
#include "bloch.h"
#include "nrutil.h"
#include "stdio_x.h"
```

```
FILE *fp_save,*fp_read,*fp_HB;
float _huge Bloch_matrix[1+maxNdetune][NVAR+1];
float     CTsignal[maxNN2+1];    /*** defining declaration ***/
/*** The following values can be modified in set_values routine. ***/
float   Bloch0[NVAR+1];
double  tstart[maxNpulse+1],tstop[maxNpulse+1],chirpBW[maxNpulse+1];
double  omegaR[maxNpulse+1],phi0[maxNpulse+1],detuneL[maxNpulse+1];
double  detune1,detuneN,ddetune,phiD,eps,area,chirp,gamma;
double noise,inhomo_HW;
int     Ndetune,arun,Npulse,jp,NN,NNpower;
int     calc_mode[maxNpulse+1],data_flag[maxNpulse+1];
long unsigned int data_value[maxNpulse+1];
char save_filename[20],read_filename[20]="",values_filename[20]="";
char CT_filename[20]="graphit.t",HB_filename[20]="graphit.f";
int save_flag,read_number,printout_flag;
void main(void)
{
  int brk,idetune,bit_value,j_sub;
  long unsigned int mask;
  double tau;
/*console_options.pause_atexit=0;
  console_options.nrows=37;                    */
/**    Quick C for Windows additions:                **/
/*_wabout("Optical Bloch Equation Solver\n\tby\nW. R. Babbitt"); */
/**    end of Quick C Additions    **/
/*Bloch_matrix=matrix(1,maxNdetune,0,NVAR);
  CTsignal=vector(1,maxNN2);  */
  /*** set initial values ***/
  init_values();
  /*** set parameters for first run ***/
  brk = set_values();
  while((brk == 0))
  {
    /*** Open save file ***/
    if(save_flag == 1)
      if(( fp_save = fopen(save_filename,"a") )== NULL)
      {
        printf("save_file %s can not be opened\n",save_filename);
        brk=1;
        save_flag = 0;
      }
    /*** Initialize Bloch_matrix ***/
    brk += init_bloch();
    /*** print run parameters to screen and file ***/
    printf("%6s %6s %6s %6s %8s %4s %7s %7s %7s\n",
      "Bl0[1]","Bl0[2]","Bl0[3]","phiD","eps","arun","Ndetune","detune1","detuneN");
    printf("%6.3f %6.3f %6.3f %6.3f %8g %4d %7d %7.3f %7.3f\n",
      Bloch0[1],Bloch0[2],Bloch0[3],(phiD/PI),eps,arun,Ndetune,(detune1/PI2),
      (detuneN/PI2));
    printf("%6s= %d\n","Npulse",Npulse);
    if ((save_flag == 1)&&(brk==0))
    {
      fprintf(fp_save,"\n \n");
      fprintf(fp_save,"%6s\t%6s\t%6s\t%6s\t%8s\t%4s\t%7s\t%7s\t%7s\t%7s\n","Bl0[1]",
        "Bl0[2]","Bl0[3]","phiD","eps","arun","Ndetune","detune1","detuneN",
        "Inho_HW");
      fprintf(fp_save,"%6.3f\t%6.3f\t%6.3f\t%6.3f\t%8g\t%4d\t%7d\t%7.4f\t%7.4f\t%7.4f\n",
        Bloch0[1],Bloch0[2],Bloch0[3],(phiD/PI),eps,arun,Ndetune,(detune1/PI2),
    (detuneN/PI2),(inhomo_HW/PI2));
  fprintf(fp_save,"%6s\n","Npulse");
  fprintf(fp_save,"%6d\n",Npulse);
}
/*** integrate pulses ***/
for(jp=1;((jp <= Npulse)&&(brk==0)); jp++)
{
  if (data_flag[jp]==0) /*** Single pulse ***/
  {
    brk=integrate_pulse(Bloch_matrix,fp_save,jp,calc_mode[jp],eps,omegaR[jp],
      tstart[jp],tstop[jp],detuneL[jp],phi0[jp],chirpBW[jp],gamma,
      detune1,ddetune,Ndetune,printout_flag,save_flag);
  }
  else  /*** Data stream (multiple subpulses) ***/
  {
    for(j_sub=0,mask=(long unsigned)1;((mask<=data_value[jp])&&(brk==0));j_sub++,mask<<=1)
    {
      tau=tstop[jp]-tstart[jp];
      bit_value=(mask&data_value[jp])?1:0;
      printf("Pulse#%d is a Data Stream  data_value=%lu mask=%lu  bit_value=%1d\n",
        jp,data_value[jp],mask,bit_value);
      switch(data_flag[jp])
      {
        case 1:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,
            (omegaR[jp]*(double)bit_value),(tstart[jp]+(double)j_sub*tau),
            (tstart[jp]+(double)(j_sub+1)*tau),detuneL[jp],phi0[jp],chirpBW[jp],
            0.0,detune1,ddetune,Ndetune,0,0);
          break;
        }
        case 2:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,omegaR[jp],
            (tstart[jp]+(double)j_sub*tau),(tstart[jp]+(double)(j_sub+1)*tau),
            detuneL[jp],(phi0[jp]+PI*(double)bit_value),chirpBW[jp],0.0,
            detune1,ddetune,Ndetune,0,0);
          break;
        }
        case 3:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,
            (omegaR[jp]*(double)bit_value),(tstart[jp]+(double)j_sub*tau),
            (tstart[jp]+(double)(j_sub+1)*tau),detuneL[jp],
            (phi0[jp]+0.5*PI*(double)j_sub),chirpBW[jp],0.0,
            detune1,ddetune,Ndetune,0,0);
          break;
        }
        case 4:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,omegaR[jp]*
            exp(((tstart[jp]+(double)j_sub*2.0*tau)-20)*gamma*2.0),
            (tstart[jp]+(double)j_sub*2.0*tau),
            (tstart[jp]+((double)j_sub*2.0+1.0)*tau),
            detuneL[jp],(phi0[jp]+PI*(double)bit_value),chirpBW[jp],0.0,
            detune1,ddetune,Ndetune,0,0);
        for(idetune=1;idetune <= Ndetune; idetune++){
            Bloch_matrix[idetune][2]*=exp(-2.0*tau*gamma);
            Bloch_matrix[idetune][3]*=exp(-2.0*tau*gamma);
         }
          break;
        }
        case 5:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,
             (omegaR[jp]*(double)bit_value),(tstart[jp]+(double)j_sub*2.0*tau),
             (tstart[jp]+((double)j_sub*2.0+1.0)*tau),detuneL[jp],phi0[jp],
             chirpBW[jp],0.0,detune1,ddetune,Ndetune,0,0);
          break;
        case 6:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,omegaR[jp]*
             exp(-((tstart[jp]+(double)j_sub*2.0*tau)-35)*gamma*2.0),
             (tstart[jp]+(double)j_sub*2.0*tau),
             (tstart[jp]+((double)j_sub*2.0+1.0)*tau),
             detuneL[jp],(phi0[jp]+PI*(double)bit_value),chirpBW[jp],0.0,
             detune1,ddetune,Ndetune,0,0);
         for(idetune=1;idetune <= Ndetune; idetune++){
            Bloch_matrix[idetune][2]*=exp(-2.0*tau*gamma);
            Bloch_matrix[idetune][3]*=exp(-2.0*tau*gamma);
         }
          break;
        }
        case 7:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,omegaR[jp],
             (tstart[jp]+(double)j_sub*2.0*tau),
             (tstart[jp]+((double)j_sub*2.0+1.0)*tau),
             detuneL[jp],(phi0[jp]+PI*(double)bit_value),chirpBW[jp],0.0,
             detune1,ddetune,Ndetune,0,0);
         for(idetune=1;idetune <= Ndetune; idetune++){
            Bloch_matrix[idetune][2]*=exp(-2.0*tau*gamma);
            Bloch_matrix[idetune][3]*=exp(-2.0*tau*gamma);
         }
          break;
        }
        case 8:
        {
          brk=integrate_pulse(Bloch_matrix,fp_save,jp,2,eps,omegaR[jp]*
             exp(((tstart[jp]+(double)j_sub*2.0*tau)-20)*gamma*2.0),
             (tstart[jp]+(double)j_sub*2.0*tau),
             (tstart[jp]+((double)j_sub*2.0+1.0)*tau),
             detuneL[jp],(phi0[jp]+PI*(double)bit_value),chirpBW[jp],0.0,
             detune1,ddetune,Ndetune,0,0);
         for(idetune=1;idetune <= Ndetune; idetune++){
            Bloch_matrix[idetune][2]*=exp(-2.0*tau*gamma);
            Bloch_matrix[idetune][3]*=exp(-2.0*tau*gamma);
         }
          break;
        }
        }
    }  /*** end of switch ***/
  }  /*** end of for(j_sub) ***/
/*printf("\n");*/
if (printout_flag == 1)
   printf("%4s %7s %9s %10s %10s %9s\n","idet",
            "detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]");
        if (save_flag == 1)
        {
          area= omegaR[jp]*tau;
          chirp=(fabs(chirpBW[jp])>TINY)?(chirpBW[jp]/tau):0.0;
          fprintf(fp_save,"%6s\t%4s\t%4s\t%10s\n","pulse#","calc","data","data_value");
          fprintf(fp_save,"%6d\t%4d\t%4d\t%10lu\n",jp,2,data_flag[jp],data_value[jp]);
          fprintf(fp_save,"%12s\t%8s\t%8s\t%10s\t%7s\t%5s\t%7s\t%10s \n",
            "omegaR","tstart","tstop","area","detuneL","phi0","chirpBW","chirp");
          fprintf(fp_save,"%12.6g\t%8.3f\t%8.3f\t%10.3g\t%7.3f\t%5.3f\t%7.3f\t%10.3g\n",
            (omegaR[jp]/PI2),tstart[jp],tstop[jp],(area/PI),(detuneL[jp]/PI2),
            (phi0[jp]/PI),(chirpBW[jp]/PI2),(chirp/PI2));
          fprintf(fp_save,"%4s\t%7s\t%9s\t%10s\t%10s\t%9s\t%4s\t%4s\n",
            "idet","detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]","junk","junk");
        }
        for(idetune=1;((idetune <= Ndetune)&&(brk==0)); idetune++)
        {
          if (printout_flag == 1)
            printf("%4d %7.4f %9.7f %10.7f %10.7f %9.7f\n",idetune,
              ((detune1+(double)(idetune-1)*ddetune)/PI2),Bloch_matrix[idetune][1],
              Bloch_matrix[idetune][2],Bloch_matrix[idetune][3]);
          if ((save_flag == 1)&(printout_flag != 2))
            fprintf(fp_save,"%d\t%.4g\t%g\t%g\t%g\t%g\t%d\t%d\n",
              idetune,((detune1+(double)(idetune-1)*ddetune)/PI2),
              Bloch_matrix[idetune][1],Bloch_matrix[idetune][2],
              Bloch_matrix[idetune][3],Bloch_matrix[idetune][0],0,0);
        }
      }/*** end of if else ***/
    } /*** end of for(jp) ***/
    /*** Store Holeburning Spectrum out for graphit to graph ***/
    if(brk==0)
    {
      if(( fp_HB = fopen(HB_filename,"w") )== NULL)
      {
        printf("HB_File %s can not be opened",HB_filename);
        brk=1;
      }
      else
      {
        for(idetune=1;(idetune <= Ndetune); idetune++)
        {
          fprintf(fp_HB,"%g\t%g\n",
            ((detune1+(double)(idetune-1)*ddetune)/PI2),
            Bloch_matrix[idetune][1]*Bloch_matrix[idetune][0]);
        }
        fclose(fp_HB);
      }
    }
    /*** Do FFT if no break and Ndetune > 16 and frequency center is 0 GHz ***/
    if ((brk == 0)&&(Ndetune>16)&&(SQR(detuneN+detune1)<TINY))
      brk=FFT_output(Bloch_matrix,CTsignal,detune1,detuneN,inhomo_HW,noise,
        phiD,NN,CT_filename,0); /*gamma)  Moe 12/19/96;*/
    if (save_flag == 1)
      fclose(fp_save);
   if ((arun <= 1)||(brk!=0))
   {
    printf("Done****************************\n");
/*  printf("Done%c%c%c\n",'\7','\7','\7');  */
```

```
       brk=set_values();   /*** set parameters for next run ***/
      }
      else
       brk=autorun();/*** perform arun commands ***/
       /*** Close output file after each run ***/
    }    /*** end while(brk == 0)  ***/
/* free_matrix(Bloch_matrix,1,maxNdetune,0,NVAR);
   free_vector(CTsignal,1,maxNN2);              */
}
int autorun(void)
{
   int brk=0;
   jp=1;
   arun--;      /*** decrement number of automatic runs ***/
/* omegaR[1] *= 0.1;
   area= omegaR[1]*(tstop[1]-tstart[1]);*/
   return(brk);
}
int init_bloch(void)
{
      int brk=0,idetune,ivar,iline,cbuff,i_occur;
      float  dummy1,dummy2,dummy3,dummy0;
      char ibuff[256];
      /*** Initialize Bloch_matrix ***/
      for(idetune=1;idetune <= Ndetune; idetune++)
         for(ivar=0;ivar <= NVAR; ivar++)
            Bloch_matrix[idetune][ivar]=Bloch0[ivar];
      if(read_number>0)
      {
         if (save_flag == 1)
            fclose(fp_save);
         if((fp_read = fopen(read_filename,"r") )!= NULL)
         {
            printf("Searching file %s for %d occurrence of idetune string.\n",
               read_filename,read_number);
            for(i_occur=1;i_occur<=read_number;i_occur++)
               for(iline=1;((fgets(ibuff,100,fp_read)!=NULL)&&!(sscanf(ibuff,"idet%c",
                  &cbuff)));iline++);
            if (!(sscanf(ibuff,"idet%c",&cbuff)))
            {
               if (i_occur=1)
                  printf("The string idetune was not found in file %s \n",read_filename);
               else
                  printf("Only %d occurrences of idetune string were found\n",i_occur);
               read_number=0;
               brk=1;
            }
            else
            {
               printf("Reading file %s starting at line %d\n",read_filename,iline+1);
               fgets(ibuff,255,fp_read);
               sscanf(ibuff,"%d %lf %f %f %f %f",&idetune,&detune1,&dummy1,&dummy2,
                  &dummy3,&dummy0);
               Bloch_matrix[1][1]=dummy1;
               Bloch_matrix[1][2]=dummy2;
               Bloch_matrix[1][3]=dummy3;
               Bloch_matrix[1][0]=dummy0;
               detune1 *= PI2;
```

```
        while ( (fgets(ibuff,100,fp_read)!=NULL) &&(sscanf(ibuff,"%d",&idetune)) )
        {
          sscanf(ibuff,"%d %lf %f %f %f %f",&idetune,&detuneN,&dummy1,&dummy2,
            &dummy3,&dummy0);
          if (idetune!=-1)
          {
            Bloch_matrix[idetune][1]=dummy1;
            Bloch_matrix[idetune][2]=dummy2;
            Bloch_matrix[idetune][3]=dummy3;
            Bloch_matrix[idetune][0]=dummy0;
          }
        }
        if (idetune==-1)
        {
          printf("Error in reading read file. idetune = -1. Cancelling read_file.\n");
          detuneN=ddetune*((double)(Ndetune-1))+detune1;
          read_number=0;
          brk=1;
        }
        else
        {
          detuneN *= PI2;
          Ndetune=idetune;
          NN=Ndetune-1;
          if(Ndetune != 1) ddetune=(detuneN-detune1)/((double)(Ndetune-1));
          printf("Ndetune=%7d  detune1/PI2=%7.3f  detuneN/PI2=%7.3f\n",Ndetune,
            (detune1/PI2),(detuneN/PI2));
          if (printout_flag == 1)
          {
            printf("%4s %7s %9s %10s %10s %9s\n","idet",
              "detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]");
            for(idetune=1;idetune <= Ndetune; idetune++)
              printf("%4d %7.4f%9.7f%10.7f%10.7f%9.7f\n",idetune,
                ((detune1+(double)(idetune-1)*ddetune)/PI2),Bloch_matrix[idetune][1],
                Bloch_matrix[idetune][2],Bloch_matrix[idetune][3],
                Bloch_matrix[idetune][0]);
          }
        }
      }
      fclose(fp_read);
    }
    else
    {
      printf("Read_File %s can not be opened",read_filename);
      read_number=0;
      brk=1;
    }
    if (save_flag == 1)
    {
      if(( fp_save = fopen(save_filename,"a") )== NULL)
      {
        printf("save_file %s can not be reopened",save_filename);
        brk=1;
        save_flag = 0;
      }
      else
      {
        if(read_number>0)
```

```
                {
                  fprintf(fp_save,"Read_file= %s   Occurrence= %d   Starting line= %d\n",
                    read_filename,read_number,iline+1);
                  fprintf(fp_save,"%4s\t%7s\t%9s\t%10s\t%10s\t%10s\n",
                    "idet","detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]");
                  for(idetune=1;idetune <= Ndetune; idetune++)
                    fprintf(fp_save,"%4d\t%7.4f\t%9.7f\t%10.7f\t%10.7f\t%10.7f\t%d\t%d\n",
                      idetune,((detune1+(double)(idetune-1)*ddetune)/PI2),
                      Bloch_matrix[idetune][1],Bloch_matrix[idetune][2],
                      Bloch_matrix[idetune][3],Bloch_matrix[idetune][0],0,0);
                }
              }
            }
          }
          return(brk);
      }
      void init_values(void)
      {
        arun=0;      /*** arun = number of times program is to run automatically ***/
        printout_flag=0;
        read_number=0;
        read_filename[0]='\0';
        phiD=0;
        NNpower =1;
        NN = 1 << NNpower;
        Ndetune=NN+1;    /*** number of detunings to evaluate = 1+ 2^NNpower ***/
        detune1 = -2.000*PI2;  /*** first detuning in Grad/sec ***/
        detuneN =  2.000*PI2;  /*** last detuning in Grad/sec ***/
        inhomo_HW = 0.0;
        noise = 0.0;
        if(Ndetune != 1) ddetune=(detuneN-detune1)/((double)(Ndetune-1));
        Bloch0[1]=1.0;   /*** starting value of rho00 ***/
        Bloch0[2]=0.0;   /*** starting value of real part of rho01 ***/
        Bloch0[3]=0.0;   /*** starting value of imag part of rho01 ***/
        Bloch0[0]=1.0;   /*** starting value of population ***/
        eps=1.0e-9;      /*** Set error parrameters ***/
        Npulse=1;    /*** number of pulses to integrate ***/
        for(jp=1;jp <= maxNpulse; jp++)
        {
          data_flag[jp]=0;
          data_value[jp]=(long unsigned)0;
          tstart[jp]=5.0;      /*** tstart,tstop in nanoseconds ***/
          tstop[jp]=15.0;
          omegaR[jp]=0.025*PI2;  /*** omegaR in gigarads/sec ***/
          phi0[jp]= 0.0*PI;   /*** phi0 in radians ***/
          chirpBW[jp]= 0.000*PI2;       /*** chirpBW in gigarads/sec ***/
          detuneL[jp]= 0.000*PI2;       /*** detune in gigarads/sec ***/
          calc_mode[jp]=2;  /*** 0 for Runge-Kutta,1 for Bulirsch-Stoer, 2 for analytic ***/
        }
        tstart[2]=45.0;
        tstop[2]=55.0;
        omegaR[2]=0.05*PI2;
        gamma=0.0;
        jp=1;        /*** current pulse number ***/
        chirp=chirpBW[jp]/(tstop[jp] - tstart[jp]); /*** chirp in gigarads/sec/ns ***/
        area= omegaR[jp]*(tstop[jp]-tstart[jp]); /*** pulse area of current [jp] pulse ***/
      }
      int set_values(void)
```

```
{
  int brk=0,jjp,j_sub,jp_insert,bit_value;
  unsigned long int mask;
  char choice,ibuff[256];
  int nchoice;
  char choice1,choice2;
  char schoice[80];
  double ytemp;
  jp=1;
  arun=0;
  choice= '\0';
  while (choice != 'i')
  {
    area=omegaR[jp]*(tstop[jp]-tstart[jp]);
    chirp=(fabs(chirpBW[jp]>TINY))?(chirpBW[jp]/(tstop[jp] - tstart[jp])):0.0;
    if((choice == 'v')||(choice=='V'))
    {
      printf("%6s %6s %6s %6s %8s %4s %7s %7s %7s %7s\n","B0[1]","B0[2]",
        "B0[3]","phiD","eps","arun","Ndetune","detune1","detuneN","Inho_HW");
      printf("%6s %6s %6s %6s %8s %4s %7s %7s %7s %7s\n",
        "rho00","rho01r","rho01i","{/PI}"," "," "," "," ","{GHz}","{GHz}","{GHz}");
      printf("%6.3f%6.3f%6.3f%6.3f%8.1e %4d %7d %7.3f%7.3f%7.3f\n",
        Bloch0[1],Bloch0[2],Bloch0[3],(phiD/PI),eps,arun,Ndetune,(detune1/PI2),
        (detuneN/PI2),(inhomo_HW/PI2));
      printf("%6s %6s %6s %6s %8s %4s %7s %7s %7s %7s\n",
        "Z,z","X,x","Y,y","P","E,e","U,u","N,n","D,d","D,d","W,w");
      printf(" \n");
      printf("%4s %4s %4s %4s %10s %4s %4s%16s%16s\n","Npls","pls#","mode","data",
        "data_value","prnt","rd#","read_filename","save_filename");
      printf("%4s %4s %4s %4s %10s %4s %4s%16s%16s\n",
        " "," ","0-2","0,1","","0,1","","","");
      printf("%4d %4d %4d %4d %10lu %4d %4d%16s%16s\n",Npulse,jp,calc_mode[jp],
        data_flag[jp],data_value[jp],printout_flag,read_number,read_filename,
        save_filename);
      printf("%4s %4s %4s %4s %10s %4s %4s%16s%16s\n","K,k","J,j","M,m","H0h1","H,h",
        "F,f","r","R->param,r->rho","S,s");
      printf("\n");
      printf("%8s %8s %8s %10s %7s %5s %7s %10s %8s\n",
        "omegaR","tstart","tstop","area","detuneL","phi0","chirpBW","chirp","Homo LW");
      printf("%8s %8s %8s %10s %7s %5s %7s %10s %8s\n",
        "{GHz}","{nsec}","{nsec}","{/PI}","{GHz}","{/PI}","{GHz}","{GHz/ns}","{GHz}");
      printf("%8.6g %8.3f%8.3f%10.3g %7.3f%5.3f%7.3f%10.3g %8e\n",
        (omegaR[jp]/PI2),tstart[jp],tstop[jp],(area/PI),(detuneL[jp]/PI2),
        (phi0[jp]/PI),(chirpBW[jp]/PI2),(chirp/PI2),(gamma/PI));
      printf("%8s %8s %8s %10s %7s %5s %7s %10s %8s\n",
        "O,o","T,t","T,t","A->t,a->o","L,l","p","B,b->L","C->t,c->b","G,g");
      if(choice=='V')
      {
        printf("\nRecap of all %d pulses\n",Npulse);
        for(jjp=1;jjp<=Npulse;jjp++)
        {
          printf("%4d %4d %4d %4d %10lu\n",Npulse,jjp,calc_mode[jjp],data_flag[jjp],
            data_value[jjp]);
          printf("%12.6g %8.3f%8.3f%10.3g  %7.3f%5.3f%7.3f%10.3g\n",
            (omegaR[jjp]/PI2),tstart[jjp],tstop[jjp],
            ((omegaR[jjp]*(tstop[jjp]-tstart[jjp]))/PI),(detuneL[jjp]/PI2),
            (phi0[jjp]/PI),(chirpBW[jjp]/PI2),
            (((fabs(chirpBW[jjp])>TINY)?(chirpBW[jjp]/(tstop[jjp] - tstart[jjp])):0.0)/PI2));
        }
      }
      printf(" \n");
    }
    printf("Enter choice (v=variable list,q=quit,i=integrate)(currently, jp=%d): ",jp);
    choice = (char)getche();
    printf("\n");
    switch(choice)
    {
      case 'A':
        area /= PI;
        do
        {
          printf("input area (in units of PI)(>=0) (modifies tstop): ");
          sscanf(gets(ibuff),"%lf",&area);
        }while (area<0.0);
        area *= PI;
        tstop[jp]=tstart[jp] + ((omegaR[jp]>TINY)?(area/omegaR[jp]):TINY);
        break;
      case 'a':
        area /= PI;
        do
        {
          printf("input area (in units of PI)(>=0) (modifies rabi frequency): ");
          sscanf(gets(ibuff),"%lf",&area);
        }while (area<0.0);
        area *= PI;
        omegaR[jp] = area/(tstop[jp]-tstart[jp]);
        break;
      case 'B':
        chirpBW[jp] /= PI2;
        printf("input chirpBW (in gigahertz)(Doesn't modify detuneL): ");
        sscanf(gets(ibuff),"%lf",&chirpBW[jp]);
        chirpBW[jp] *= PI2;
        chirp=(fabs(chirpBW[jp]>TINY))?(chirpBW[jp]/(tstop[jp] - tstart[jp])):0.0;
        break;
      case 'b':
        chirpBW[jp] /= PI2;
        printf("input chirpBW (in gigahertz)(also modifies detuneL): ");
        sscanf(gets(ibuff),"%lf",&chirpBW[jp]);
        chirpBW[jp] *= PI2;
        chirp=(fabs(chirpBW[jp])>TINY)?(chirpBW[jp]/(tstop[jp] - tstart[jp])):0.0;
        detuneL[jp] = -0.5*chirpBW[jp];
        break;
      case 'C':
        chirp /= PI2;
        do
        {
          printf("input chirp (in gigahertz/nsec)(modifies tstop[jp]): ");
          sscanf(gets(ibuff),"%lf",&chirp);
        }while (chirp==0.0);
        chirp *= PI2;
        tstop[jp] = (chirpBW[jp]/chirp)+tstart[jp];
        break;
      case 'c':
        chirp /= PI2;
        printf("input chirp (in gigahertz/nsec)(modifies chirpBW[jp]): ");
        sscanf(gets(ibuff),"%lf",&chirp);
        chirp *= PI2;
        chirpBW[jp]=chirp*(tstop[jp]-tstart[jp]);
        break;
      case 'D':
      case 'd':
        if(read_number==0)
        {
          do
          {
            detune1 /= PI2;
            printf("input detune1: ");
            sscanf(gets(ibuff),"%lf",&detune1);
            detune1 *= PI2;
            detuneN /= PI2;
            printf("input detuneN (must be > detune1): ");
            sscanf(gets(ibuff),"%lf",&detuneN);
            detuneN *= PI2;
          }    while (detune1 >= detuneN);
          if((Ndetune>16)&&(SQR(detuneN+detune1)<TINY))
            printf("FFT info: delta_t = %6.3f nsec      Maximum t = %8.3f nsec",
              PI2/(detuneN-detune1),((double)(NN-1)*PI2/(detuneN-detune1)));
        }
        else
          printf("Can't set detunings when read_number =%d.\n",read_number);
        break;
      case 'E':
      case 'e':
        printf("input eps: ");
        sscanf(gets(ibuff),"%lf",&eps);
        break;
      case 'F':
      case 'f':
        do
        {
          printf("Frequency Information Printout Options:\n0) File Printout Only\n");
          printf("1) File and screen printout\n2) no printout to file or screen\n");
          printf("Input desired option: ");
          sscanf(gets(ibuff),"%d",&printout_flag);
        }while ((printout_flag!=0)&&(printout_flag!=1)&&(printout_flag!=2));
        break;
      case 'G':
      case 'g':
        if (gamma==0.0)
        {
          printf("Currently T2 is infinite and gamma =0\n");
          ytemp=1.0e9;
        }
        else
        {
          ytemp = 1.0/gamma;
          printf("Currently T2 = %lf nsec \n",ytemp);
        }
        printf("input homogeneous decay time,T2 ( >=1e9 means inf): ");
        sscanf(gets(ibuff),"%lf",&ytemp);
        if (ytemp>=1.0e9)
        {
          gamma=0.0;
          printf("T2 is now infinite and gamma =0\n");
```

```
    }
    else
    {
       gamma = 1.0/ytemp;
       printf("T2 = %lf nsec, gamma = %lf, Homo LW= %lf GHz\n",ytemp,gamma,(gamma/PI));
    }
    break;
  case 'H':
    data_flag[jp]=0;
    data_value[jp]=(long unsigned)0;
    break;
  case 'h':
    do
    {
      printf("modulation mode options: \n");
      printf("0)Cancel 1)AM NRZ 2)PM 3)AM w/phase rotation 4)PM w/RZ & up ramp & decay\n");
      printf("5)AM RZ 6) PM w/RZ & down ramp & decay  7)PM w/RZ w/decay:\n");
      printf("input modulation mode: ");
      sscanf(gets(ibuff),"%d",&data_flag[jp]);
    }while ((data_flag[jp]<0)&&(data_flag[jp]>7));
    printf("input data_value (0 to 4294967295): ");
    sscanf(gets(ibuff),"%lu",&data_value[jp]);
    for(j_sub=0,mask=(long unsigned)1;((mask<=data_value[jp])&&(brk==0));j_sub++,mask<<=1)
    {
      bit_value=(mask&data_value[jp])?1:0;
      printf("%1d",bit_value);
    }
    printf(" is the binary sequence for %lu",data_value[jp]);
    calc_mode[jp]=2;
    break;
  case 'I':
    do
    {
      printf("Input # of pulse to insert(>0)/deleted(<0)/escape(0):");
      sscanf(gets(ibuff),"%d",&jp_insert);
      if (abs(jp_insert)>Npulse)
         printf("abs(jp_insert) > Npulse = %d\n",Npulse);
    }while (abs(jp_insert)>Npulse);
    if (jp_insert<0)
    {
      for(jjp=-jp_insert;jjp < Npulse; jjp++)
      {
        data_flag[jjp]=data_flag[jjp+1];
        data_value[jjp]=data_value[jjp+1];
        tstart[jjp]=tstart[jjp+1];
        tstop[jjp]=tstop[jjp+1];
        omegaR[jjp]=omegaR[jjp+1];
        phi0[jjp]=phi0[jjp+1];
        chirpBW[jjp]=chirpBW[jjp+1];
        detuneL[jjp]=detuneL[jjp+1];
        calc_mode[jjp]=calc_mode[jjp+1];
      }
      printf("Pulse# %d to pulse# %d have been shifted down one.\n",
        (-jp_insert),Npulse);
      printf("The old pulse# %d has been deleted.\n",(-jp_insert));
      Npulse--;
      break;
    }
    if (Npulse == maxNpulse)
    {
      printf("Npulse =maxNpulse= %8d.  Can't insert more pulses. \n",maxNpulse);
      break;
    }
    if (jp_insert>0)
    {
      for(jjp=Npulse;jjp >= jp_insert; jjp--)
      {
        data_flag[jjp+1]=data_flag[jjp];
        data_value[jjp+1]=data_value[jjp];
        tstart[jjp+1]=tstart[jjp];
        tstop[jjp+1]=tstop[jjp];
        omegaR[jjp+1]=omegaR[jjp];
        phi0[jjp+1]=phi0[jjp];
        chirpBW[jjp+1]=chirpBW[jjp];
        detuneL[jjp+1]=detuneL[jjp];
        calc_mode[jjp+1]=calc_mode[jjp];
      }
      printf("Pulse# %d to pulse# %d have been shifted up one.\n",jp_insert,Npulse);
      printf("Pulse# %d is a duplicate of the old pulse# %d.\n",jp_insert,jp_insert);
      Npulse++;
      break;
    }
    break;
  case 'i':
    printf("Integrating (press b to break)");
    break;
  case 'J':
  case 'j':
    jp = Npulse +1;
    while (jp > Npulse)
    {
      printf("input pulse# (=jp): ");
      sscanf(gets(ibuff),"%d",&jp);
      if (jp > Npulse) printf("Npulse= %8d \n",Npulse);
    }
    area=omegaR[jp]*(tstop[jp]-tstart[jp]);
    chirp=(fabs(chirpBW[jp]>TINY))?(chirpBW[jp]/(tstop[jp] - tstart[jp])):0.0;
    break;
  case 'K':
  case 'k':
    Npulse = maxNpulse +1;
    while ((Npulse > maxNpulse)||(Npulse<1))
    {
      printf("input Npulse: ");
      sscanf(gets(ibuff),"%d",&Npulse);
      if (Npulse > maxNpulse) printf("maxNpulse= %8d \n",maxNpulse);
      jp=1;
    }
    break;
  case 'L':
  case 'l':
    detuneL[jp] /= PI2;
    printf("input detuneL: ");
    sscanf(gets(ibuff),"%lf",&detuneL[jp]);
    detuneL[jp] *= PI2;
    break;
    case 'M':
    case 'm':
      if(data_flag[jp]==0)
      {
        do
        {
          printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
            "0=Runge-Kutta","1=Bulirsch-Stoer (constant amp)",
            "2=Square Pulse","3=Coherence Loss","4=Gating Pulse",
            "5=Bulirsch-Stoer with decay","6=Bulirsch-Stoer (shaped pulse)",
            "7=gating and noise");
          printf("input calc_mode: ");
          sscanf(gets(ibuff),"%d",&calc_mode[jp]);
        }while ((calc_mode[jp]<0)||(calc_mode[jp]>7));
        if (calc_mode[jp]==5)
          printf("Currently Homo LW= %lf GHz \n",gamma/PI);
        if ((calc_mode[jp]==3)||(calc_mode[jp]==4)||(calc_mode[jp]==7))
        {
          tstart[jp]=(jp>1)?tstop[jp-1]:0.0;
          tstop[jp]=tstart[jp]+0.001;
          omegaR[jp]=0.0;
          chirpBW[jp]=0.0;
        }
      }
      else
        printf("Can't change calc_mode unless data_flag=0.\n");
      break;
   case 'N':
   case 'n':
     if(read_number==0)
     {
       do
       {
         if ((Ndetune>16)&&(SQR(detuneN+detune1)<TINY))
           printf("NNpower=%d  Ndetune=%d  delta_t=%6.3fnsec  Maximum t=%8.3fnsec\n",
             NNpower,Ndetune,PI2/(detuneN-detune1),
             ((double)(NN-1)*PI2/(detuneN-detune1)));
         printf("input NNpower (Ndetune=2^NNpower +1)");
         sscanf(gets(ibuff),"%d",&NNpower);
         NN = 1 << NNpower;
         Ndetune=NN+1;
         if ((Ndetune>maxNdetune)||(NNpower>14))
           printf("maxNdetune= %8d \n",maxNdetune);
       }while ((Ndetune>maxNdetune)||(NNpower>14));
       printf("Ndetune = %d\n",Ndetune);
       if ((Ndetune>16)&&(SQR(detuneN+detune1)<TINY))
         printf("FFT info: delta_t = %6.3f nsec      Maximum t = %8.3f nsec\n",
           PI2/(detuneN-detune1),((double)(NN-1)*PI2/(detuneN-detune1)));
     }
     else
       printf("Can't set Ndetune when read_number =%d.\n",read_number);
     break;
  case 'O':
  case 'o':
    omegaR[jp] /= PI2;
    do
    {
      printf("input omegaR (in gigaghertz)(>=0): ");
```

```
          sscanf(gets(ibuff),"%lf",&omegaR[jp]);
        } while (omegaR[jp]<0.0);
        omegaR[jp] *= PI2;
        area=omegaR[jp]*(tstop[jp]-tstart[jp]);
        break;
      case 'P':
        phiD /= PI;
        printf("input phase of coherent detection phiD (in units of PI): ");
        sscanf(gets(ibuff),"%lf",&phiD);
        phiD *= PI;
        break;
      case 'p':
        phi0[jp] /= PI;
        printf("input phi0 (in units of PI): ");
        sscanf(gets(ibuff),"%lf",&phi0[jp]);
        phi0[jp] *= PI;
        break;
      case 'Q':
      case 'q':
        choice='i';
        brk=1;
        break;
      case 'R':
        brk=read_values();
        choice='V';
        brk=0;
        break;
      case 'r':
        printf("Input file from which to read in Bloch values (CR=none): ");
        gets(read_filename);
        if (read_filename[0] == '\0')
        {
          printf("No entry (setting read_number=0):\n ");
          read_number=0;            /*** No output file ***/
        }
        else
        {
          read_number=0;
          printf("Input occurance number of desired bloch values (read_number>0): ");
          sscanf(gets(ibuff),"%d",&read_number);
        }
        if(read_number>0)
          printf("Will read file = %s  for occurance #%d \n",read_filename,read_number);
        else
        {
          read_number=0;
          printf("Occurance number =0.  Cancelling read_file.\n ");
          read_filename[0] = '\0';
        }
        break;
      case 'S':
      case 's':
      printf("Input the save file name (type CR for none):\n");
        gets(save_filename);
        if (save_filename[0] == (char)NULL)
          save_flag=0; /*** No output file ***/
        else
        {
          save_flag = 1;
          printf("output file = %s \n",save_filename);
        }
        break;
      case 'T':
      case 't':
        do
        {
          printf("input tstart (in nanseconds): ");
          sscanf(gets(ibuff),"%lf",&tstart[jp]);
          printf("input tstop: ");
          sscanf(gets(ibuff),"%lf",&tstop[jp]);
        } while(tstart[jp]>=tstop[jp]);
        area=omegaR[jp]*(tstop[jp]-tstart[jp]);
        chirp=(fabs(chirpBW[jp])>TINY)?(chirpBW[jp]/(tstop[jp] - tstart[jp])):0.0;
        break;
      case 'U':
      case 'u':
        printf("input arun (1=autorun): ");
        sscanf(gets(ibuff),"%d",&arun);
        break;
      case 'W':
        inhomo_HW /= PI2;
        printf("input inhomo_HW (0:flat, >0:Guassian HW1/eM,<0:Lorentzian HWHM)(GHz): ");
        sscanf(gets(ibuff),"%lf",&inhomo_HW);
        inhomo_HW *= PI2;
        if (SQR(inhomo_HW)<TINY)
          printf("Flat inhomogeneous profile \n");
        if (inhomo_HW>TINY)
          printf("Guassian inhomogeneous profile with HW1/eM = %lf",inhomo_HW/PI2);
        if (inhomo_HW<(-TINY))
          printf("Lorentzian inhomogeneous profile with HWHM = %lf",(-inhomo_HW/PI2));
        break;
      case 'w':
        noise *= PI2;
        printf("noise=%lf per GHz . Input new noise per GHz: ",noise);
        sscanf(gets(ibuff),"%lf",&noise);
        noise /= PI2;
        break;
      case 'X':
      case 'x':
        printf("input Bloch0[2] (rho01r): ");
        sscanf(gets(ibuff),"%f",&Bloch0[2]);
        break;
      case 'Y':
      case 'y':
        printf("input Bloch0[3] (rho01i): ");
        sscanf(gets(ibuff),"%f",&Bloch0[3]);
        break;
      case 'Z':
      case 'z':
        printf("input Bloch0[1] (rho00): ");
        sscanf(gets(ibuff),"%f",&Bloch0[1]);
        break;
    }
    if(Ndetune != 1) ddetune=(detuneN-detune1)/((double)(Ndetune-1));
```

```
  if((chirpBW[jp]!=0)&&((calc_mode[jp]!=5)&&(calc_mode[jp]!=6)&&(calc_mode[jp]!=1)&&(calc_mode[jp]!
=0)))
     {
       printf("calc_mode must equal 0 or 1 or 5 or 6 when chirp != 0.  Setting calc_mode=1\n");
       calc_mode[jp]=1;
     }
     printf("\n");
  }
  return(brk);
}
int read_values(void)
{
     int brk=0,iline,cbuff,i_occur,jjp,values_number;
     char ibuff[256];
     FILE  *fp_values;
     /*** Reads in parameter values from a file***/
     printf("input file from which to read parameter values: ");
     gets(values_filename);
     if (values_filename[0] != '\0')
     {
       do
       {
         printf("input run number of desired values (values_number>0): ");
         sscanf(gets(ibuff),"%d",&values_number);
       }while (values_number<1);
       if((fp_values = fopen(values_filename,"r") )!= NULL)
       {
         printf("Searching file %s for %d occurrence of Bl0[1] string.\n",
           values_filename,values_number);
         for(i_occur=1;i_occur<=values_number;i_occur++)
           for(iline=1;((fgets(ibuff,100,fp_values)!=NULL)&&!(sscanf(ibuff,"Bl0[1]%c",
             &cbuff)));iline++);
         if(!(sscanf(ibuff,"Bl0[1]%c",&cbuff)))
         {
           if (i_occur=1)
             printf("The string Bl0[1] was not found in file %s \n",values_filename);
           else
             printf("Only %d occurrences of Bl0[1] string were found\n",i_occur);
           brk=1;
         }
         else
         {
           printf("Reading file %s starting at line %d\n",values_filename,iline+1);
           fgets(ibuff,255,fp_values);
           sscanf(ibuff,"%f %f %f %lf %lf %d %d %lf %lf %lf\n",&Bloch0[1],&Bloch0[2],
             &Bloch0[3],&phiD,&eps,&arun,&Ndetune,&detune1,&detuneN,&inhomo_HW);
           NN=Ndetune-1;
           detune1 *= PI2;
           detuneN *= PI2;
           inhomo_HW *= PI2;
           arun=0;
           fgets(ibuff,255,fp_values);
           fgets(ibuff,255,fp_values);
           sscanf(ibuff,"%d",&Npulse);
           if ((Npulse > maxNpulse)||(Npulse<1))
           {
             printf("Npulse = %d is <1 or >maxNpulse= %d \n",Npulse,maxNpulse);
```

```
            brk=2;
          }
          for(jjp=1;((jjp<=Npulse)&&(brk==0));jjp++)
          {
            for(iline=1;((fgets(ibuff,255,fp_values)!=NULL)&&!(sscanf(ibuff,"pulse#%c",
              &cbuff)));iline++);
            if((sscanf(ibuff,"pulse#%c",&cbuff)))
            {
              fgets(ibuff,255,fp_values);
              sscanf(ibuff,"%d %d %d %ul",&jp,&calc_mode[jjp],&data_flag[jjp],
                &data_value[jjp]);
              if (jp==jjp)
              {
                fgets(ibuff,255,fp_values);
                fgets(ibuff,255,fp_values);
                sscanf(ibuff,"%lf %lf %lf %lf %lf %lf %lf %lf",&omegaR[jp],&tstart[jp],
                  &tstop[jp],&area,&detuneL[jp],&phi0[jp],&chirpBW[jp],&chirp);
                omegaR[jp]*=PI2;
                detuneL[jp]*=PI2;
                phi0[jp]*=PI;
                chirpBW[jp]*=PI2;
                area*=PI;
                chirp*=PI2;
              }
              else
              {
                printf("Error:jp=%d does not equal jjp=%d.\n",jp,jjp);
                brk=2;
              }
            }
            else
            {
              printf("Error: string pulse# not found for jjp=%d.\n",jjp);
              brk=2;
            }
          }
          fgets(ibuff,255,fp_values);
          if((!(sscanf(ibuff,"idet%c",&cbuff)))&&(brk==0))
          {
            printf("Warning: The next line doesn't contain the string idetune.\n");
          }
        }
        fclose(fp_values);
      }
      else
      {
        printf("Values_file %s can not be opened\n",values_filename);
        brk=1;
      }
      switch(brk)
      {
        case 0:
          printf("Read successful.\n");
          break;
        case 1:
          printf("Read unsuccessful.\n");
          break;
        case 2:
            printf("Read unsuccessful. Values were altered.  Reinitializing values:\n");
            save_flag = 0;
            init_values();
            break;
        }
    }
    jp=1;
    return(brk);
}
```

OBE_BS.C

```
#define IMAX 11
#define NUSE 7
#define SHRINK 0.95
#define GROW 1.2
#define NVAR 3
#include <math.h>
#include <stdio.h>
#include "bloch.h"
#include "nrutil.h"
double d[NVAR+1][NUSE+1],x[IMAX+1]; /* defining declaration */
void bsstep(double y[], double dydx[], double *x, double htry, double eps, double yscal[],
  double *hdid, double *hnext, void (*derivs )(), double A, double B, double C,
  double gamma, double omegaR)
{
  int i,j;
  double xsav,xest,h,errmax,temp;
  double ysav[NVAR+1],dysav[NVAR+1],yseq[NVAR+1],yerr[NVAR+1];
  static int nseq[IMAX+1]={0,2,4,6,8,12,16,24,32,48,64,96};
  void mmid(),rzextr(),nrerror();
  h=htry;
  xsav=(*x);
  for (i=1;i<=NVAR;i++) {
    ysav[i]=y[i];
    dysav[i]=dydx[i];
    /*** Take square root of yscal (WRB 1/10/92) ***/
    yscal[i] = exp(0.5*log(yscal[i]));
  }
  for (;;) {
    for (i=1;i<=IMAX;i++) {
      mmid(ysav,dysav,xsav,h,nseq[i],yseq,derivs,A,B,C,gamma,omegaR);
      xest=(temp=h/nseq[i],temp*temp);
      rzextr(i,xest,yseq,y,yerr);
      errmax=0.0;
      for (j=1;j<=NVAR;j++)
        if (errmax < fabs(yerr[j]/yscal[j]))
          errmax=fabs(yerr[j]/yscal[j]);
      errmax /= eps;
      if (errmax < 1.0) {
        *x += h;
        *hdid=h;
        *hnext = i==NUSE? h*SHRINK : i==NUSE-1?
          h*GROW : (h*nseq[NUSE-1])/nseq[i];
        return;
      }
    }
    h *= 0.25;
    for (i=1;i<=(IMAX-NUSE)/2;i++) h /= 2.0;
```

```
      if ((*x+h) == (*x)) nrerror("Step size underflow in BSSTEP");
    }
  }
void mmid(double y[], double dydx[], double xs, double htot, int nstep, double yout[],
   void (*derivs )(), double A, double B, double C, double gamma, double omegaR)
{
   int n,i;
   double x,swap,h2,h,ym[NVAR+1],yn[NVAR+1];
   h=htot/nstep;
   for (i=1;i<=NVAR;i++) {
      ym[i]=y[i];
      yn[i]=y[i]+h*dydx[i];
   }
   x=xs+h;
   (*derivs)(x,yn,yout,A,B,C,gamma,omegaR);
   h2=2.0*h;
   for (n=2;n<=nstep;n++) {
      for (i=1;i<=NVAR;i++) {
         swap=ym[i]+h2*yout[i];
         ym[i]=yn[i];
         yn[i]=swap;
      }
      x += h;
      (*derivs)(x,yn,yout,A,B,C,gamma,omegaR);
   }
   for (i=1;i<=NVAR;i++)
      yout[i]=0.5*(ym[i]+yn[i]+h*yout[i]);
}
```

OBE_FFT.C

```
/*
"four1() replaces data[] by its discrete Fourier transform, if isign=1; or
replaces data[] by nn times its inverse discrete Fourier transform, if
isign=-1. data[] is a complex array of length nn, input as a real array
data[1...2*nn]. nn MUST be an integer power of 2. (This is not checked!)
Reference: Numerical Recipes, pp. 407-412.
*/
#define  PI2 6.28318530717958
#define TINY   1.0e-18 /*** definition of a tiny value ***/
#define SQR(a) ((a)*(a))
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
#define PI 3.14159265358979323846
#include <math.h>
#include <stdio.h>
#include "bloch.h"
#include "random.h"
static long idum2=(-1L);
static int idum1=-1;
static int idumg = -13;
int FFT_output(float _huge Bloch_matrix[][NVAR+1], float CTsignal[], double detune1, double detuneN,
   double inhomo_HW, double noise, double phiD, int NN, char CT_filename[],double gamma)
{
      int isign,ift,brk,idetune,NNhalf,NN2;
      double delta_t,lineshape,ddetune,sigmax,sigmaxt;
      FILE *fp_CT;
      if (noise < 0.0)
         idumg = (-13);
      printf("Performing FFT on output and storing it in %s .\n",CT_filename);
```

```
delta_t=PI2/(detuneN-detune1);
/*** convert density matrix to output electric field ***/
NNhalf=NN >> 1;
NN2=NN << 1;
ddetune=(detuneN-detune1)/((double)NN);
for(idetune=NNhalf+1,ift=1;idetune<=NN+1;idetune++,ift +=2)
{
  lineshape=1.0;
  if (inhomo_HW>TINY)
    lineshape=exp(-SQR((detune1+ddetune*(double)(idetune-1))/inhomo_HW));
  if (inhomo_HW<(-TINY))
    lineshape=1.0/(1.0+SQR((detune1+ddetune*(double)(idetune-1))/inhomo_HW));
  if (fabs(lineshape)<TINY) lineshape = 0.0;
  lineshape *= 1 + noise*ddetune*gasdev(&idumg);
  if (fabs(lineshape)<TINY) lineshape = 0.0;
  CTsignal[ift] = (float)lineshape*Bloch_matrix[idetune][2]*Bloch_matrix[idetune][0];
  CTsignal[ift+1] = (float)lineshape*Bloch_matrix[idetune][3]*Bloch_matrix[idetune][0];
}
for(idetune=2,ift=NN+3;ift<=NN2-1;idetune++,ift +=2)
{
  lineshape=1.0;
  if (inhomo_HW>TINY)
    lineshape=exp(-SQR((detune1+ddetune*(double)(idetune-1))/inhomo_HW));
  if (inhomo_HW<(-TINY))
    lineshape=1.0/(1.0+SQR((detune1+ddetune*(double)(idetune-1))/inhomo_HW));
  if (fabs(lineshape)<TINY) lineshape = 0.0;
  lineshape *= 1 + noise*ddetune*gasdev(&idumg);
  if (fabs(lineshape)<TINY) lineshape = 0.0;
  CTsignal[ift] = (float)lineshape*Bloch_matrix[idetune][2]*Bloch_matrix[idetune][0];
  CTsignal[ift+1] = (float)lineshape*Bloch_matrix[idetune][3]*Bloch_matrix[idetune][0];
}
isign=1;
FFT(CTsignal,NN,isign);
/*** Store CTsignal out for graphit to graph ***/
if(( fp_CT = fopen(CT_filename,"w") )== NULL)
{
  printf("CT_File %s can not be opened",CT_filename);
  brk=1;
}
else
{
  sigmax =0.0;
  sigmaxt = 0.0;
  for(ift=1;ift<=NN2-1;ift +=2) {
    fprintf(fp_CT,"%g\t%.5g\t%.5g \n",
      (double)(ift-1)*0.5*delta_t,(SQR(CTsignal[ift])+SQR(CTsignal[ift+1]))*
      exp(-(((double)(ift-1)*0.5*delta_t-190)*gamma*2.0)),
      (cos(phiD)*CTsignal[ift+1]-sin(phiD)*CTsignal[ift])*
      exp(-(((double)(ift-1)*0.5*delta_t-190)*gamma)) );
    if ((double)(ift-1)*0.5*delta_t > 70.0)
     if ((SQR(CTsignal[ift])+SQR(CTsignal[ift+1]))> sigmax) {
        sigmax =(SQR(CTsignal[ift])+SQR(CTsignal[ift+1]));
        sigmaxt =(double)(ift-1)*0.5*delta_t;
      }
  }
  fclose(fp_CT);
  printf("sigmaxt = %12.5g   simmax  = %12.5g \n",sigmaxt,sigmax);
}
```

```
        return(brk);
}
```

OBE_INT.C

```
/*This program is to be used in conjunction with bloch.c for
integrating the Optical Bloch Equations.
It is based on odeint.c, except that intemediate value of the integration
are no longer stored in xp and yp.*/
#define MAXSTP 10000
#define TINY 1.0e-30
#define NVAR 3
#define  PI  3.14159265358979
#define  PI2 6.28318530717958
#include <stdlib.h>
#include <bios.h>
#include <conio.h>
#include <io.h>
#include <stdio.h>
#include <math.h>
#include "complex.h"
#include "bloch.h"
#include "nrutil.h"
#include "stdio_x.h"
static int idumg2=-13;
int integrate_pulse(float _huge Bloch_matrix[][NVAR+1], FILE *fp_save, int jpulse, int calc_mode,
   double eps, double omegaR, double tstart, double tstop, double detuneL, double phi0,
   double chirpBW, double gamma, double detune1, double ddetune, int Ndetune,
   int printout_flag, int save_flag)
{
   int nbad,nok,idetune,ivar,brk=0;
   unsigned key = 0;
   double h1,hmin,Bloch[NVAR+1],deltaA,deltaB,A,B,C,detune,detuneAL,tau,area,chirp;
     tau=tstop-tstart;
     area= omegaR*tau;
     chirp=(fabs(chirpBW)>TINY)?(chirpBW/tau):0.0;
     printf("%6s %4s\n","pulse#","calc");
     printf("%6d %4d\n",jpulse,calc_mode);
     if (calc_mode==5)
        printf("gamma = %lf\n",gamma);
     printf("%12s %8s %8s %10s %7s %5s %7s %10s \n",
        "omegaR","tstart","tstop","area","detuneL","phi0","chirpBW","chirp");
     printf("%12.6g %8.3f %8.3f %10.3g %7.3f %5.3f %7.3f %10.3g\n",
        (omegaR/PI2),tstart,tstop,(area/PI),(detuneL/PI2),
        (phi0/PI),(chirpBW/PI2),(chirp/PI2));
     if (printout_flag == 1)
        printf("%4s %7s %9s %10s %10s %9s %4s %4s\n","idet",
           "detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]","nok","nbad");
     if (save_flag == 1)
     {
        fprintf(fp_save,"%6s\t%4s\t%4s\t%10s\n","pulse#","calc","data","data_value");
        fprintf(fp_save,"%6d\t%4d\t%4d\t%10d\n",jpulse,calc_mode,0,0);
        fprintf(fp_save,"%12s\t%8s\t%8s\t%10s\t%7s\t%5s\t%7s\t%10s \n",
           "omegaR","tstart","tstop","area","detuneL","phi0","chirpBW","chirp");
        fprintf(fp_save,"%12.6g\t%8.3f\t%8.3f\t%10.3g\t%7.3f\t%5.3f\t%7.3f\t%10.3g\n",
          (omegaR/PI2),tstart,tstop,(area/PI),(detuneL/PI2),
          (phi0/PI),(chirpBW/PI2),(chirp/PI2));
        fprintf(fp_save,"%4s\t%7s\t%9s\t%10s\t%10s\t%9s\t%4s\t%4s\n",
           "idet","detune","Bloch[1]","Bloch[2]","Bloch[3]","Bloch[0]","nok","nbad");
```

```
  }
  detune=detune1;
  detuneAL= detune1 - detuneL;
  B= -detuneAL;
  C=0.5*chirp;
  A=  phi0 - detuneAL*tstart;
  deltaA = -ddetune*tstart;
  deltaB = -ddetune;
  switch(calc_mode)
  {
    case -1:  /*** do nothing ***/
      break;
    case 0:     /*** Runge-Kutta integration ***/
      hmin=0.0;
      h1=0.1;
      break;
    case 1: /*** Bulirsch-Stoer integration with constant amplitude***/
      hmin=0.0;
      h1=2.0;
      break;
    case 2:   /*** Analytic solution ***/
      nok = 1;
      nbad = 0;
      break;
    case 3:   /*** coherence loss ***/
      break;
    case 4:   /*** gating pulse ***/
      break;
    case 5:     /*** Bulirsch-Stoer integration with decay***/
      hmin=0.0;
      h1=2.0;
      break;
    case 6:     /*** Bulirsch-Stoer integration with shaped pulses***/
      hmin=0.0;
      h1=2.0;
      break;
    case 7:   /*** gating pulse and noise***/
      break;
  }
 for(idetune=1;((idetune <= Ndetune)&&(brk==0)); idetune++)
 {
   /*** set Bloch to old value of Bloch_matrix ***/
   for(ivar=0;ivar <= NVAR; ivar++) Bloch[ivar]=(double) Bloch_matrix[idetune][ivar];
   switch(calc_mode)
   {
     case -1:  /*** do nothing ***/
       break;
     case 0:     /*** Runge-Kutta integration ***/
       if (omegaR>TINY)
          OBEint(Bloch,0.0,tau,eps,h1,hmin,&nok,&nbad,nodecay,A,B,C,0.0,
            omegaR,rkqc);
       break;
     case 1:     /*** Bulirsch-Stoer integration with constant amp***/
       if (omegaR>TINY)
          OBEint(Bloch,0.0,tau,eps,h1,hmin,&nok,&nbad,nodecay,A,B,C,gamma,
            omegaR,bsstep); /* for constant omegaRt gamma<=> 0.0 moe 12/19/96  */
       break;
     case 2:  /*** Analytic solution ***/
```

```
            if (omegaR>TINY)
              SquarePulse(Bloch,tau,A,-B,C,omegaR);
            break;
          case 3:   /*** coherence loss ***/
            Bloch[2]=0.0;
            Bloch[3]=0.0;
            break;
          case 4:   /*** gating pulse ***/
            Bloch[2]=0.0;
            Bloch[3]=0.0;
            Bloch[0] *= Bloch[1];
            Bloch[1]=1.0;
            break;
          case 5:       /*** Bulirsch-Stoer integration with decay***/
            if (omegaR>TINY)
              OBEint(Bloch,0.0,tau,eps,h1,hmin,&nok,&nbad,decay,A,B,C,gamma,
                omegaR,bsstep); /*assumes constant amplitude pulse*/
            break;
          case 6:       /*** Bulirsch-Stoer integration with shaped pulse***/
            if (omegaR>TINY)
              OBEint(Bloch,0.0,tau,eps,h1,hmin,&nok,&nbad,nodecay,A,B,C,(PI2/tau),
                omegaR,bsstep);     /*for shaped pulses*/
            break;
          case 7:   /*** gating pulse and noise***/
            Bloch[2]=0.0;
            Bloch[3]=0.0;
            Bloch[0] *= Bloch[1];
            Bloch[0] += omegaR*ddetune*gasdev(&idumg2);
            Bloch[1]=1.0;
            break;
        }
        /*** store new value in Bloch_matrix ***/
        for(ivar=0;ivar <= NVAR; ivar++)
        {
/*        modf(100000000.0*Bloch[ivar],&Bloch[ivar]);***1e-8 accuracy***
          Bloch_matrix[idetune][ivar] = Bloch[ivar]*0.00000001; */
          Bloch_matrix[idetune][ivar] = (float)Bloch[ivar];
        }
        if (printout_flag == 1)
          printf("%4d %7.4f %9.7f %10.7f %10.7f %9.7f %4d %4d\n",idetune,(detune/PI2),
            Bloch_matrix[idetune][1],Bloch_matrix[idetune][2],
            Bloch_matrix[idetune][3],Bloch_matrix[idetune][0],nok,nbad);
        if ((save_flag == 1)&((printout_flag != 2)||(idetune==1)))
          fprintf(fp_save,"%d\t%.4g\t%g\t%g\t%g\t%g\t%d\t%d\n",
            idetune,(detune/PI2),Bloch_matrix[idetune][1],Bloch_matrix[idetune][2],
            Bloch_matrix[idetune][3],Bloch_matrix[idetune][0],nok,nbad);
        A += deltaA;
        B += deltaB;
        detune += ddetune;
        if (_bios_keybrd(_NKEYBRD_READY))
        {
          key = _bios_keybrd( _NKEYBRD_READ   );
          if ((key & 0x00ff) == 'b')
          {
            brk = 1;
            printf("break\n");
            if (save_flag == 1)
              fprintf(fp_save,"%6d\n",-1);
            }
          }
        } /*** end for(ident=...) ***/
      return(brk);
    }
    void OBEint(double ystart[], double x1, double x2, double eps, double h1, double hmin,
      int *nok, int *nbad,void (*derivs )(), double A, double B,
      double C, double gamma, double omegaR, void (*stepper )())
    {
      int nstp,i;
      double x,hnext,hdid,h;
      double yscal[NVAR+1],y[NVAR+1],dydx[NVAR+1];
      void nrerror();
      x=x1;
      h=(x2 > x1) ? fabs(h1) : -fabs(h1);
      *nok = (*nbad) = 0;
      for (i=1;i<=NVAR;i++) y[i]=ystart[i];
      for (nstp=1;nstp<=MAXSTP;nstp++) {
        (*derivs)(x,y,dydx,A,B,C,gamma,omegaR);
        if ((x+h-x2)*(x+h-x1) > 0.0) h=x2-x;
        for (i=1;i<=NVAR;i++)
          /*** Use absolute accuracy scaled to step size (WRB 1/10/92) ***/
          yscal[i]=fabs(h/(x2-x1));
        (*stepper)(y,dydx,&x,h,eps,yscal,&hdid,&hnext,derivs,A,B,C,gamma,omegaR);
        if (hdid == h) ++(*nok); else ++(*nbad);
        if ((x-x2)*(x2-x1) >= 0.0) {
          for (i=1;i<=NVAR;i++) ystart[i]=y[i];
          return;
        }
        if (fabs(hnext) <= hmin) nrerror("Step size too small in OBEINT");
        h=hnext;
      }
      nrerror("Too many steps in routine OBEINT");
      return;
    }
    void nodecay(double t, double Bloch[], double dBlochdt[], double A, double B, double C,
      double gamma, double omegaR)
    {
      double phit,omegaRt,a1,a2,a3;
      if(gamma==0.0)  omegaRt = omegaR;
       else  omegaRt = omegaR*(1.0-cos(PI2*t*gamma)*cos(PI2*t*gamma));        /*PI2 et al
      moe 12/19/96*/
      phit= A + B*t + C*t*t;
      a1= omegaRt*sin(phit);
      a2= omegaRt*cos(phit);
      a3= 0.5 - Bloch[1];
      dBlochdt[1]= (a2*Bloch[3] - a1*Bloch[2]);
      dBlochdt[2]= -a1*a3;
      dBlochdt[3]= a2*a3;
    }
    void decay(double t, double Bloch[], double dBlochdt[], double A, double B, double C,
      double gamma, double omegaR)
    {
      double phit,omegaRt,a1,a2,a3;
      phit= A + B*t + C*t*t;
      a1= omegaR*sin(phit);
      a2= omegaR*cos(phit);
      a3= 0.5 - Bloch[1];
```

```
  dBlochdt[1]= (a2*Bloch[3] - a1*Bloch[2]);
  dBlochdt[2]= -a1*a3-gamma*Bloch[2];
  dBlochdt[3]= a2*a3-gamma*Bloch[3];
}
```

OBE_RK.C

```
#define PGROW -0.20
#define PSHRNK -0.25
#define FCOR 0.06666666                    /* 1.0/15.0 */
#define SAFETY 0.9
#define ERRCON 6.0e-4
#define NVAR 3
#include <math.h>
#include <stdio.h>
#include "nrutil.h"
#include "bloch.h"
void rkqc(double y[], double dydx[], double *x, double htry, double eps, double yscal[],
  double *hdid, double *hnext, void (*derivs )(),
  double A, double B, double C, double gamma, double omegaR)
{
  int i;
  double xsav,hh,h,temp,errmax;
  double dysav[NVAR+1],ysav[NVAR+1],ytemp[NVAR+1];
  void rk4(),nrerror();
  xsav=(*x);
  for (i=1;i<=NVAR;i++) {
    ysav[i]=y[i];
    dysav[i]=dydx[i];
    /*** Take square root of yscal (WRB 1/10/92) ***/
    yscal[i] = sqrt(yscal[i])*16.0;
  }
  h=htry;
  for (;;) {
    hh=0.5*h;
    rk4(ysav,dysav,xsav,hh,ytemp,derivs,A,B,C,gamma,omegaR);
    *x=xsav+hh;
    (*derivs)(*x,ytemp,dydx,A,B,C,gamma,omegaR);
    rk4(ytemp,dydx,*x,hh,y,derivs,A,B,C,gamma,omegaR);
    *x=xsav+h;
    if (*x == xsav) nrerror("Step size too small in routine RKQC");
    rk4(ysav,dysav,xsav,h,ytemp,derivs,A,B,C,gamma,omegaR);
    errmax=0.0;
    for (i=1;i<=NVAR;i++) {
      ytemp[i]=y[i]-ytemp[i];
      temp=fabs(ytemp[i]/yscal[i]);
      if (errmax < temp) errmax=temp;
    }
    errmax /= eps;
    if (errmax <= 1.0) {
      *hdid=h;
      *hnext=(errmax > ERRCON ?
        SAFETY*h*exp(PGROW*log(errmax)) : 4.0*h);
      break;
    }
    h=SAFETY*h*exp(PSHRNK*log(errmax));
  }
  for (i=1;i<=NVAR;i++) y[i] += ytemp[i]*FCOR;
}
```

```
void rk4(double y[], double dydx[], double x, double h, double yout[],
  void (*derivs )(), double A, double B, double C, double gamma,
  double omegaR)
{
  int i;
  double xh,hh,h6,dym[NVAR+1],dyt[NVAR+1],yt[NVAR+1];
  hh=h*0.5;
  h6=h/6.0;
  xh=x+hh;
  for (i=1;i<=NVAR;i++) yt[i]=y[i]+hh*dydx[i];
  (*derivs)(xh,yt,dyt,A,B,C,gamma,omegaR);
  for (i=1;i<=NVAR;i++) yt[i]=y[i]+hh*dyt[i];
  (*derivs)(xh,yt,dym,A,B,C,gamma,omegaR);
  for (i=1;i<=NVAR;i++) {
    yt[i]=y[i]+h*dym[i];
    dym[i] += dyt[i];
  }
  (*derivs)(x+h,yt,dyt,A,B,C,gamma,omegaR);
  for (i=1;i<=NVAR;i++)
    yout[i]=y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);
}
```

OBE_SQ.C

```
#include <stdio.h>
#include <math.h>
#include "complex.h"
#include "bloch.h"
#include "nrutil.h"
#define SQR(a) ((a)*(a))
void SquarePulse(double y[], double tau, double phi, double delta, double C, double rabi)
{
double chi,theta,theta_2;
fcomplex rho01_start,rho01_final,rho00_start,rho00_final,Cphase,Ctemp1,Ctemp2,Ctemp3;
void nrerror();
if( C!=0.0)
  nrerror("Chirp does not equal zero");
else
  {
  rho00_start = Complex(y[1],0.0);
  rho01_start = Complex(y[2],y[3]);
  chi = sqrt(SQR(delta) + SQR(rabi));
  theta = chi*tau;
  theta_2 = 0.5*theta;
  Cphase = Complex(0,phi);
  Ctemp1=Cmul(
        Complex((SQR(cos(theta_2))-SQR(delta*sin(theta_2)/chi)),
            (delta*sin(theta)/chi)),
        rho01_start);
  Ctemp2=Cmul(
        RCmul(
            SQR(rabi*sin(theta_2)/chi),
            Cexp(RCmul(2.0,Cphase))),
        Conjg(rho01_start) );
  Ctemp3=Cmul(
        Complex(
            delta*rabi*SQR(sin(theta_2)/chi), -
            rabi*0.5*sin(theta)/chi) ,
        Cexp(Cphase) );
```

```
    rho01_final=Cmul(
            Cadd(
              Cadd(
                Ctemp1,
                Ctemp2 ),
              RCmul(
                (2.0*rho00_start.r - 1.0),
                Ctemp3)),
            Cexp(Complex(0.0,-delta*tau) ) );
    rho00_final.r=rho00_start.r + (1.0-2.0*rho00_start.r)*SQR(rabi*sin(theta_2)/chi)
      +(rabi*sin(theta)/chi)*(cos(phi)*rho01_start.i - sin(phi)*rho01_start.r)
      +2.0*rabi*delta*SQR(sin(theta_2)/chi)*
        (sin(phi)*rho01_start.i + cos(phi)*rho01_start.r);
    y[2]=rho01_final.r;
    y[3]=rho01_final.i;
    y[1]=rho00_final.r;
    }
    return;
}
```